





## Method for the automatic generation of a controller

**Patent number:** DE19513801  
**Publication date:** 1996-10-17  
**Inventor:** WINKELMANN KLAUS DR ING (DE)  
**Applicant:** SIEMENS AG (DE)  
**Classification:**  
- **international:** G05B19/05  
- **european:** G05B19/042P  
**Application number:** DE19951013801 19950411  
**Priority number(s):** DE19951013801 19950411

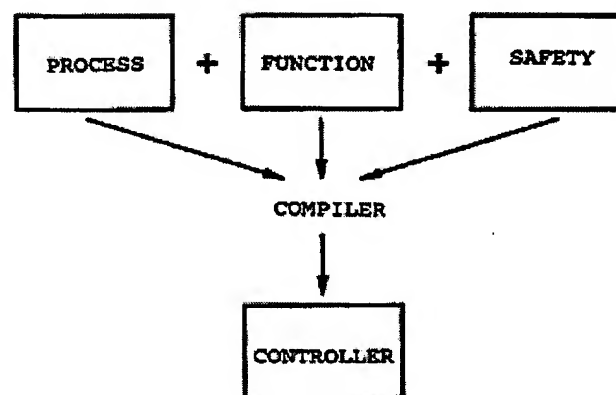
**Also published as:**

 WO9632667 (A1)  
 EP0820610 (A1)  
 US6047278 (A1)  
 EP0820610 (B1)

Report a data error here

**Abstract of DE19513801**

A process allows a control to be generated from an application-related formal specification. The process allows controls to be generated that satisfy specific safety conditions so that the safety conditions are respected during control generation. Constructs for specifying sequential and parallel processes allow the exclusive description of only the functionally desired aspects, so that a clear distinction between safety and function is achieved.



Data supplied from the esp@cenet database - Worldwide

BEST AVAILABLE COPY



① BUNDESREPUBLIK  
DEUTSCHLAND



DEUTSCHES  
PATENTAMT

⑫ **Offenlegungsschrift**  
⑩ **DE 195 13 801 A 1**

⑤ Int. Cl.<sup>6</sup>:  
**G 05 B 19/05**

⑳ Aktenzeichen: 195 13 801.5  
㉑ Anmeldetag: 11. 4. 95  
㉒ Offenlegungstag: 17. 10. 96

DE 195 13 801 A 1

㉓ Anmelder:  
Siemens AG, 80333 München, DE

㉔ Erfinder:  
Winkelmann, Klaus, Dr.-Ing., 85521 Riemerling, DE

㉕ Entgegenhaltungen:  
DE 37 43 438 A1  
JÖRNS, C., LITZ, L., BERGOLD, S.: Automatische  
Erzeugung von SPS-Programmen auf der Basis von  
Petri-Netzen. In: atp-Auto- matisierungstechnische  
Praxis 37 (1995) 3, S.10-14;

Prüfungsantrag gem. § 44 PatG ist gestellt

㉖ Verfahren zur automatischen Erzeugung einer Steuerung

㉗ Es wird ein Verfahren beschrieben, das die Generierung  
einer Steuerung aus einer anwendungsnahen, formalen  
Spezifikation unterstützt.

Das Verfahren ermöglicht die Erzeugung von Steuerungen,  
die spezifizierte Sicherheitsbedingungen erfüllen, und zwar  
so, daß der Generierungsprozeß deren Einhaltung garantiert.  
Konstrukte zur Spezifikation sequentieller und paralleler  
Abläufe sind vorgesehen, die es erlauben, nur gerade die  
funktional gewünschten Aspekte zu beschreiben, so daß  
eine klare Trennung von Sicherheit und Funktion möglich ist.

DE 195 13 801 A 1

Die folgenden Angaben sind den vom Anmelder eingereichten Unterlagen entnommen

## Beschreibung

Die rationelle Entwicklung zuverlässiger Software für reaktive Systeme, wie es Steuerungen aller Art sind, ist heute vorrangiges Ziel. Fig. 1 zeigt ein Prinzipbild eines Systems bestehend aus einer Steuerung und einem zu steuernden Prozeß. Die Steuerung muß so ausgeführt sein, daß der Prozeß nur zulässige Zustände einnimmt. Der Steuerung werden an den Eingängen Eingangswerte, z. B. Sensorwerte, zugeführt, die im folgenden Eingangszustände genannt werden. An den Ausgängen gibt die Steuerung Ausgangswerte, im folgenden Ausgangszustände genannt, ab, die den Prozeß so beeinflussen, daß er nur die zulässigen Zustände annehmen kann.

Bei der Programmierung derartiger speicherprogrammierbarer Steuerungen sind jeweils folgende Vorgaben zu beachten:

- die Funktionsweise der zu steuernden Geräte oder Prozesse — d. h. was bewirkt jeder einzelne Ausgangszustand der Steuerung, und wie sind die Sensorwerte, die der Steuerung geliefert werden, zu interpretieren.
- die von der Steuerung zu realisierende Funktion; z. B. Bewegungsabläufe.
- ggf. vorhandene einschränkende Sicherheitsbedingungen; z. B. sind bestimmte Kombinationen von Zuständen, oder bestimmte geometrische Konfigurationen zu vermeiden, bestimmte Aktoren dürfen nur aktiviert werden, wenn zugehörige Vorbedingungen (Verriegelungen) erfüllt sind etc.

Diese Vorgaben liegen jeweils in unterschiedlicher, meist informeller Form vor. Das technische Problem besteht nun darin, eine Steuerung so zu programmieren, daß sie den Vorgaben entspricht.

Im Gegensatz zu regelungstechnischen Aufgaben, bei denen die Theorie abhängig von Prozeßmodellen bestimmte (Standard-) Regelungsalgorithmen kennt, müssen diskrete Steuerungsaufgaben jeweils neu programmiert werden.

Die Programmierung speicherprogrammierbarer Steuerungen geschieht heute in der Weise, daß der Programmierer die logischen Verknüpfungen festlegt, nach denen die Ausgangszustände sich aus den Eingangszuständen und den Programzuständen bestimmen.

Zur Programmierung von Abläufen existieren entsprechende Konstrukte von Programmiersprachen, um sog. Schrittketten und parallele Abläufe zu beschreiben. In jedem Fall ist es aber Sache des Programmierers, die Einhaltung der Funktion und der Sicherheitsbedingungen zu garantieren. M.a. W., der Zusammenhang zwischen dem Programm und den oben genannten Vorgaben ist nicht explizit dokumentiert.

Ein Problem bei diesem Vorgehen ist die Fehleranfälligkeit, denn schon bei mittelgroßen Systemen ist die Vielfalt der möglichen Systemzustände praktisch unüberschaubar groß; so werden nahezu immer Fälle übersehen, und bei der Inbetriebnahme müssen mit erheblichem Aufwand Programme umgeschrieben werden.

Noch gravierender ist das Problem, daß jedes Umschreiben, sei es wegen eines Fehlers, wegen geänderter Umgebungsbedingungen oder wegen neuer Anforderungen, die ganze Software betrifft — d. h. es gibt keinen Mechanismus, der die Auswirkungen einer Änderung lokal hält oder auch nur die von einer Änderung betroffenen Stellen markiert.

Der Erfindung liegt die Aufgabe zugrunde, aus einer anwendungsnahen Spezifikation einer Steuerungsaufgabe eine Steuerung zu erzeugen. Diese Aufgabe wird gemäß den Merkmalen von Patentanspruch 1 gelöst.

Weiterbildungen der Erfindung ergeben sich aus den abhängigen Patentansprüchen.

Die Erfindung wird anhand eines Ausführungsbeispiels erläutert, unter Verwendung einer Spezifikationsprache, die CSLxt genannt werden soll.

## 1. Grundsätzliche Darstellung der Erfindung

## 1.1 Grundkonzepte von CSLxt

Die Erfindung geht davon aus, daß der Entwurfsprozeß für reaktive Systeme immer mit den folgenden Komponenten beginnt (Fig. 2):

- Beschreibung der zu steuernden Prozesse,
- Beschreibung der gewünschten Funktion des Gesamtsystems, und
- Sicherheitsanforderungen.

CSLxt umfaßt zum einen Sprachmittel, um die drei Komponenten der Spezifikation zu beschreiben, zum anderen die zur Generierung der Steuerung nötigen Verfahren.

Die tragenden Grundelemente des Ansatzes sind:

- die Spezifikation mit dem Konzept der endlichen Automaten,
- die Unterscheidung zwischen beeinflussbaren und nichtbeeinflussbaren Zustandsgrößen, und
- die Konstruktion einer Steuerung durch Iterationsverfahren.

Dem Verfahren liegt die Vorstellung zugrunde, daß auf jedes Ereignis im Prozeß, d. h. jede Änderung der Sensorwerte, eine Reaktion der Steuerung, d. h. eine Änderung der Aktor-Stellungen (Ausgangs-Werte) erfolgt. Das Zusammenwirken von Steuerung und Prozeß wird als ein Zwei-Personen-Spiel aufgefaßt, wobei die Steuerung ein bestimmtes Ziel verfolgt — nämlich Sicherheit und Funktion zu garantieren —, während der Prozeß als "Gegenspieler" dieses Ziel im ungünstigsten Fall zu verhindern sucht, jedenfalls aber nicht kooperiert.

Nun wird das Verhalten des Prozesses (bzw. der Prozesse) als gegeben angenommen, die durch Sensorwerte beschriebenen Zustandsgrößen sind nicht (direkt) beeinflussbar. Dagegen sind die Aktoren von der Steuerung beeinflussbar. Das Verhalten der Steuerung — d. h. hier die Übergangsfunktion des Automaten, der die Steuerung beschreibt, wird aus der Spezifikation des Ziels synthetisiert.

Dieser gesamte Generierungsprozeß läuft wie folgt ab:

1. Definition eines nicht-deterministischen Automaten, der alle physikalisch möglichen Verhaltensweisen beschreibt.
2. Beschreibung der erlaubten Zustands-Übergänge.
3. Einschränkung des durch 1. und 2. beschriebenen Automaten auf einen, der die Sicherheitseigenschaften erfüllt.
4. Einschränkung dieses Automaten auf einen, der die Funktion erfüllt.

Die letzten beiden Konstruktionsschritte machen intensiven Gebrauch von Iterationsverfahren im Zustandsraum des Gesamtsystems. Dies kann verstanden werden als ein Vorausschauen der Steuerung auf die möglichen "Spielzüge". Hierin unterscheidet sich CSLxt wesentlich von den meisten anderen Spezifikationsmethoden für reaktive Systeme, die die Übergangsfunktion eines Automaten zwar auch berechnen, aber dabei immer nur einen Schritt (einen "Zug", eine Transition) betrachten.

Das Verfahren zu 3. ist gekennzeichnet durch die folgenden Arbeitsschritte:

- Es wird die (in einem zu präzisierenden Sinne) größte stabilisierbare Teilmenge der als zulässig (sicher) spezifizierten Zustandsmenge iterativ bestimmt.
- Die Übergangsfunktion wird dadurch eingeschränkt, daß nur Übergänge zu sicheren Zuständen erlaubt werden.

Das Verfahren zu 4. ist durch die folgenden Schritte gekennzeichnet:

- die Funktion wird durch Angabe bedingter Zielzustände beschrieben.
- Es wird iterativ die Menge all derjenigen Zustände bestimmt, von denen aus der Übergang in einen Zielzustand in endlich vielen Schritten erzwungen werden kann.
- Die Übergangsfunktion wird dadurch eingeschränkt, daß, wenn immer möglich, solche Übergänge ausgeführt werden, die "näher" an den Zielzustand führen. Ein Zustand a ist dabei dem Ziel näher als ein Zustand b, wenn von a aus weniger Schritte als von b aus benötigt werden, um in den Zielzustand zu gelangen.
- Bei der Berechnung dieser Zustandsmengen wird berücksichtigt, daß für das Verhalten des gesteuerten Prozesses wohldefinierte "Fairness"-Annahmen gelten, d. h. daß bestimmte Ereignisse jeweils nach einer nicht bekannten, aber endlichen Anzahl von Schritten eintreten müssen.

### 1.2 Beispiel Hubdrehtisch, Fig. 3

Zur Illustration der Erfindung dient ein einfacher Hubdrehtisch, der hier kurz beschrieben wird:  
Die Funktion des Hubdrehtisches ist es, ein Teil, das von einem Zuführband kommt, in eine Position zu heben und (um eine vertikale Achse) zu drehen, in der es dann ein Roboter greifen kann.

Demgemäß gibt es die Befehle up, down und stop für die vertikale Bewegung sowie rotate\_plus (abgekürzt rot\_plus), rot\_minus, rot\_stop für die Drehbewegung.

Es gibt zwei Lichtschranken "low" und "high" für die vertikale Position, sowie ein analoges Signal, das den Drehwinkel meldet. Für die Eingänge machen wir jedoch durchgängig folgende vereinfachende Annahme: eine Schnittstellenkomponente außerhalb der hier spezifizierten Steuerung setzt die Eingangswerte so um, daß für die beiden Dimensionen jeweils genau einer von drei möglichen Werten vorliegt, nämlich low, between\_low\_high, oder high, und rot0, between\_rot0\_rot45, oder rot45.

Damit arbeiten wir effektiv mit diskretisierten (qualitativen) Werten.

Die Positionen (low, rot0) bzw. (high, rot45) entspricht dabei der Stellung, in der ein Teil vom Zuführband aufgelegt bzw. vom Roboter weggenommen werden kann.

Folgende Sicherheitsanforderungen sind beim Hubdrehtisch zu beachten:

"In der unteren Stellung (Sensor low = on) darf der Tisch nicht rotiert werden" (weil nämlich der Tisch sonst mit dem unmittelbar angrenzenden Zuführband kollidieren würde) bzw. im vorliegenden Kontext gleichbedeutend: "In der unteren Stellung (low) ist nur der Drehwinkel 0 (rot0) zulässig" (Siehe Fig. 2, schraffierter Bereich = verbotene Zustände).

Für die Bewegung des Hubdrehtisches ergibt sich daraus, daß zwar eine gleichzeitige Bewegung nach oben und Drehung möglich ist, aber erst nachdem die Höhe between\_low\_high erreicht wurde. Diese Art Einschränkung ist typisch für viele in der Automatisierungstechnik auftretende Parallelabläufe.

### 2. Prozeßmodellierung in CSLxt

Als Voraussetzung für die Konstruktion einer sicheren und korrekten Steuerung wird eine Beschreibung der Abläufe im (ungesteuerten) Prozeß verwendet, die im folgenden erläutert wird.

## 2.1 Vorgaben zur Anwendung von CSLxt

Bei den Zustandsattributen der Steuerung sind zu unterscheiden:

- Eingangszustände oder Sensorzustände, die direkt von außen gesetzt werden können,
- Ausgangszustände, die direkt den Prozeß beeinflussen, und
- innere Zustände oder Steuerzustände.

Es trägt zur Klarheit bei, diese drei Arten auch syntaktisch in einem CSLxt-Programm zu unterscheiden. Dies kann am Beispiel des Hubdrehtisches etwa wie folgt aussehen:

```

sensor_zustaende: [
    vertical_pos: [ low, between_low_high, high ],
    rotary_pos: [ rot0, between_rot0_rot45, rot45 ]
],

ausgangs_zustaende: [
    moving_state: [ stop, up, down ],
    rotation_state: [ rot_stop, rot_plus, rot_minus ]
],

```

Anmerkung: Die Sprache CSLxt dient hier der Illustration der Konzepte, ist aber in ihrer spezifischen Ausprägung nicht Gegenstand der Anmeldung. Daher wird die Syntax nicht vollständig beschrieben.

## 2.2 Prozeßmodellierung mit Constraints

Beim Prozeßmodell geht es darum, zu beschreiben, wie sich die Ausgangszustände auf Eingangszustände auswirken, d. h. welche Übergänge der Eingangszustände bei gegebener Belegung der Ausgänge möglich sind. Insofern ist die eingeführte Unterscheidung zwischen Eingangszuständen, Ausgangszuständen und inneren Zuständen die Basis des Prozeßmodells.

Das Prozeßmodell kann als endlicher Automat beschrieben werden, der mit der Steuerung gekoppelt ist, d. h. er liest die Ausgangszustände und schreibt die Eingangszustände der Steuerung. Für die praktische Notation des Prozeßmodells wird jedoch eine Darstellung gewählt, die diese Sichtweise nicht explizit macht. Aus Benutzersicht ist das Prozeßmodell ja schlicht ein Zusammenhang von Eingangszuständen und Ausgangszuständen — die im folgenden vorgeschlagene Notation ist daher Teil einer CSLxt-Spezifikation. Dazu wird "process\_description" als neues Konstrukt eingeführt, und anhand des folgenden Beispiels beschrieben.

Ein sehr einfaches, aber praktisch häufig auftretendes Beispiel ist die Überwachung einzelner binärer Ausgänge durch direkt zugeordnete Sensoren, etwa "Ausgangszustand: Hebel öffnen, Eingangszustand: Hebel ist offen". Diese Art Zusammenhänge wird bisher nur im Kopf des Entwicklers, aber nicht formal verarbeitet — eine häufige Fehlerquelle.

Für die folgende Darstellung wählen wir ein anderes, etwas komplexeres Beispiel, das wiederum exemplarisch ist für einen häufig vorkommenden Typ von Ausgangs-Eingangs-Zusammenhängen: Ein Ausgangswert gibt die Richtung vor, in die sich ein bestimmter Eingangswert verändert bzw. verändern soll. Diese Situation liegt immer dann vor, wenn als Aktor ein Motor dient, der die Stellung einer Komponente in zwei Richtungen verändern kann.

Es wird hier angenommen, daß alle Prozeßwerte zu qualitativen Werten abstrahiert werden. Damit hat der Ausgang den Wertebereich [stop, up, down] und der Eingang z. B. drei Werte wie [low, between\_low\_high, high].

Der zugehörige Ausschnitt aus dem Prozeßmodell könnte in einer anwendungsnahen Darstellung nun wie folgt aussehen:

```

process_description: [
    qual_deriv : [vertical_pos(low, between_low_high, high),
                  moving_state(stop, up, down)]
]

```

Dies soll bedeuten, daß das Attribut moving\_state die qualitative Ableitung des Attributs vertical\_pos darstellt, wobei die genannten Werte auftreten.

Es werden natürlich neben qual\_deriv weitere Varianten gebraucht, z. B. für mehr als drei Wert-Intervalle,

evtl. für Motoren mit mehr als einer Geschwindigkeit, aber auch für Zusammenhänge anderer Art. Allerdings ist der Anwendungsbereich des hier vorgestellten qual deriv und seiner engen Verwandten größer als es die Vorstellung von einem "Motor" und einer mechanischen Position suggeriert.

Weitere Beispiele, die ebenfalls als qualitative Ableitung beschreibbar sind, umfassen:

- Ventilstellungen bzw. Pumpen als Aktoren und Füllstand-Sensoren,
- Heizung bzw. Kühlung als Aktor und Temperatur-Sensoren.

Nach dem hier vorgeschlagenen Konzept braucht also der Benutzer zur Spezifikation des Prozeßmodells die einzelnen Zustandsübergänge nicht detailliert zu beschreiben, sondern lediglich die vordefinierten qualitativen Constraints wie z. B. qual deriv verwenden. Das Prozeßmodell für den Hubdrehtisch besteht dabei aus genau zwei Constraints, nämlich dem oben genannten und einem entsprechenden für die Rotation.

Dies ist ein zusätzlicher Aufwand, der für die pure Erzeugung einer Steuerung nicht notwendig wäre und in heutigen Steuerungssprachen daher auch nicht üblich ist. Er wird aber durch den Nutzen der automatischen Generierung gerechtfertigt.

### 3. Generierung der Steuerung mit Sicherheits-Eigenschaften

Im folgenden wird ausgeführt, wie aufgrund der Spezifikation einer Sicherheitseigenschaft ein Automatenprogramm so zu transformieren ist, daß es diese Eigenschaft erfüllt. Unter Sicherheitseigenschaft wird dabei eine beliebige aussagenlogische Bedingung in den Zustandsgrößen eines Automaten verstanden.

#### 3.1 Spezifikation von Sicherheitseigenschaften

Ein Sprachkonstrukt, das zu diesem Zweck in CSLxt eingeführt wird, heißt "safety\_requirements" und kann etwa wie folgt aussehen.

Die Semantik ist einfach dadurch definiert, daß die angegebene Bedingung in jedem erreichbaren Zustand des Automaten gelten soll. Es ist dabei belanglos, ob mehrere Bedingungen oder eine Konjunktion von Bedingungen angegeben wird — unter praktischen Gesichtspunkten (debugging) ist eine Liste von (evtl. auch benannten) Bedingungen vorzuziehen.

#### safety\_requirements:

```
/*----- SAFETY REQUIREMENTS -----*/
```

Im folgenden steht V für logisches "UND", ==> für logische Implikation

```
/* Teil 1: Tisch wird nicht tiefer als "low" abgesenkt */
```

```
nicht (vertical_pos = low /\ moving_state = down)
```

```
/* ... (analog fuer die anderen Grenzen) ... */
```

```
/\
```

```
/* Teil 2: Tisch wird in der Stellung "low" nicht gedreht */
```

```
(vertical_pos = low) ==>
```

```
(rotation_state = rot_stop /\ rotary_pos = rot0)
```

```
/*-----END SAFETY REQUIREMENTS-----*/
```

#### 3.2 Lösung durch ein Minimax-Verfahren

Es wird nun das System aus Steuerung und gesteuertem Prozeß als ein Zwei-Personen-Spiel zwischen der Steuerung und dem gesteuerten Prozeß, wie folgt, betrachtet:

Abwechselnd machen die Spieler, Proc (Prozeß) und Ctrl (Steuerung) ihre Züge. Ziel des Spielers Ctrl ist es, innerhalb der als sicher definierten Zustände zu bleiben. Proc agiert blind — um den ungünstigen Fall zu beherrschen, wird aber ein "Gegenspieler" eingeführt, der versucht, in einen unsicheren Zustand zu kommen. Jeder Spieler wird die möglichen Gegenzüge berücksichtigen. Aus der Spieltheorie sind Minimax-Algorithmen als Verfahren bekannt, die für jeden Spieler in dieser Situation die optimale Strategie beschreiben.

A priori gegeben ist eine Sicherheitsanforderung (safety\_requirement) SR wie in 3.1. angegeben, und Aufgabe der Steuerung ist es, deren Einhaltung zu garantieren. Es genügt aber nicht, unmittelbare Zustandsänderungen in Zuständen außerhalb SR zu vermeiden, wie das Fig. 4 verdeutlicht, die eine graphische Darstellung der aufeinander-

der folgenden Zustände zeigt. Die Zustände von ctrl und proc sind als Kreise dargestellt, deren Abhängigkeit voneinander durch Pfeile verdeutlicht sind. Die weiß unterlegten Kreise stellen erlaubte Zustände ZW dar, die grau unterlegten Kreise stellen Zustände ZG dar, die als gefährlich zu meiden sind, weil gilt: kommt das System in einen grauen Zustand, kann nicht mehr garantiert werden, daß es nicht in einen verbotenen, illegalen Zustand ZS kommt, der durch schwarze Kreise dargestellt ist. Aufgabe des Verfahrens ist es, solche "Sackgassen" zu vermeiden.

Dabei ist zu beachten:

- Ein Zustand, in dem Proc am Zug ist, ist gefährlich, falls es einen gefährlichen Folgezustand gibt.
- Ein Zustand, in dem Ctrl am Zug ist, ist gefährlich, falls alle Folgezustände gefährlich sind.

Bei Fig. 4 hat Ctrl ausgehend vom Zustand ZW1 zwei Möglichkeiten. Wählt Ctrl den linken Ast, folgt bei Proc ein gefährlicher Zustand ZG1. ZG1 ist gefährlich, weil Proc am Zug ist und ZG2 wählen könnte. ZG2 ist gefährlich, weil keine andere Wahl als ZG3 bleibt. ZG3 ist gefährlich, weil Proc am Zug ist, durch den der Zustand ZS1 gewählt werden könnte. Die anderen Zweige können entsprechend durchlaufen werden.

Die Menge der gefährlichen Zustände auszurechnen, ist deshalb möglich, weil die gesamte Zustandsmenge endlich ist und die genannten Bedingungen immer nur Zustände hinzufügen können. Eine Iteration muß also nach endlich vielen Schritten einen Fixpunkt erreichen.

Mit "Zustandsmengen" sind im folgenden immer Mengen von Zuständen der Steuerung bezeichnet.

Für jede Zustandsmenge P bezeichnet  $acx(P)$  die Menge all der Zustände, für die in einem Schritt ein Übergang in die Menge P erzwingbar ist.

Offensichtlich besteht  $acx(P)$  aus

- allen Zuständen, in denen die Steuerung am Zug ist, sofern es wenigstens einen Folgezustand in P gibt, sowie
- allen Zuständen, in denen der Prozeß am Zug ist und für die jeder Folgezustand in P ist.

Für jede Zustandsmenge P bezeichne die "größte stabilisierbare Teilmenge von P" die größte Menge Q für die gilt

Q ist in P enthalten und  
Q ist in  $acx(Q)$  enthalten.

Es wird zunächst dargestellt, wie die größte stabilisierbare Teilmenge, einer beliebigen Zustandsmenge berechnet werden kann:

#### Verfahren zur Konstruktion der größten stabilisierbaren Teilmenge

Es sei P eine Zustandsmenge.

1. Setze  $Q_0 := P$
2. Wiederhole Schritt 3 solange bis gilt  $Q_i = Q_{i+1}$
3. Setze  $Q_{i+1} := acx(Q_i) \cap Q_i$   
( $\cap$  steht für den Durchschnitt zweier Mengen)

#### Verfahren zur Konstruktion einer sicheren Steuerung

1. Setze  $SR :=$  Menge der Zustände, die die spezifizierten safety requirements erfüllen.
2. Setze  $G :=$  größte stabilisierbare Teilmenge von SR.
3. Definiere die Funktion der Steuerung so, daß zu jedem Zustand immer ein Folgezustand gewählt wird, der in G liegt.

#### Ergebnis des Verfahrens

Falls der Anfangszustand nicht zu G gehört, dann ist die Spezifikation nicht erfüllbar. Andernfalls garantiert die Konstruktion gerade, daß für jedes Verhalten des Prozesses die Steuerung immer einen Zug (Transition) hat, die innerhalb der Zustandsmenge G bleibt.

#### 4. Spezifikation der Funktion

Eine Steuerung, die nur sicher ist, ist von geringem praktischen Wert. Es muß selbstverständlich auch garantiert sein, daß sie die gewünschte Funktion erfüllt. Im Sinne formaler Verfahren ist die erste Voraussetzung hierfür eine Spezifikation dieser Funktion.

CSLxt kennt zwei sich ergänzende Sprachmittel zur Beschreibung funktionaler Eigenschaften:

- a) Spezifikation durch explizite Angabe von Transitionen
- b) Deklarative Spezifikation durch Angabe bedingter Sollzustände.

Punkt a) erfordert kein neues Verfahren und wird daher nur kurz behandelt (Abschnitt 4.1). Punkt b) ist eine Innovation von CSLxt und wird in Abschnitt 4.2. dargestellt.

#### 4.1 Explizite Spezifikation

Dieser Abschnitt zeigt, wie die Funktion einer Steuerung beschrieben werden kann, so daß für die Einhaltung von Sicherheitseigenschaften das Verfahren aus Abschnitt 3. zum Einsatz kommen kann.

Zulässige Übergänge der Steuerung können z. B. in der gängigen Petri-Netz-Notation wie in Fig. 5 beschrieben werden. Dabei steht jeweils ein Kreis für einen Zustand der Steuerung, ein waagrecht er Strich für einen möglichen Übergang (Transition). Neben den Zuständen stehen Aktionen (Ausgangswerte) der Steuerung, neben den Transitionen Vorbedingungen, die zum Zeitpunkt der Transition erfüllt sein müssen, wie z. B. Sensor-Informationen.

Das Ergebnis des Minimax-Verfahrens besteht in diesem Beispiel darin, daß an der im Diagramm mit \*\* bezeichneten Stelle automatisch eine zusätzliche Vorbedingung eingebaut wird, die das Einhalten der Sicherheitsbedingung (erst drehen, wenn vertikale Stellung im sicheren Bereich ist) garantiert. Wird die Automatentafel durch solche explizite Transitionen beschrieben, schränkt dies die Freiheiten ein, die für die Generierung der Sicherheit gebraucht werden. Läßt die explizite Spezifikation keine solchen Freiheiten, kann die Sicherheitsgenerierung nicht mehr arbeiten. Ihr Ergebnis wäre dann

- entweder sind die Sicherheitseigenschaften ohnehin erfüllt
- oder sie können auch durch das Iterationsverfahren nicht erfüllt werden; dieses liefert dann das Ergebnis "not okay".

Daher ist bei der expliziten Spezifikation darauf zu achten, daß keine unnötigen Details festgelegt werden, sondern die Freiheiten, die aus technologischer Sicht bestehen, auch in der Spezifikation bestehen bleiben.

#### 4.2 Deklarative Spezifikation

Im folgenden wird das Sprachmittel vorgestellt, das CSLxt zur deklarativen Spezifikation der funktionalen Eigenschaften bietet. Angestrebte Vorteile sind dabei

- Komplexitätsreduktion, d. h. Vereinfachung der Spezifikation durch Konzentration auf das Wesentliche, daher
- Elimination von Fehlerquellen,
- anwendungsnahe Formulierung der funktionalen Eigenschaften.

Das Konstrukt hat die Form

function:[Prae1 — Target1, Prae2 — Target2,...]

d. h. es wird eine Liste von Paaren angegeben. Prae1, Target1 etc. stehen dabei für beliebige CSLxt-Bedingungen. Die Semantik dieses Konstrukts ist beschreibbar wie folgt:

Target1 usw. geben die Soll- oder Zielzustände an. Ein Sollzustand wird jeweils abhängig von der zugehörigen Vorbedingung Prae aktiviert. Genauer gilt für jedes Paar Prae — Target:

Solange Prae gilt, wird die Steuerung versuchen, einen Zustand herzustellen, in dem Target gilt. Dieser wird erst verlassen, wenn Prae nicht mehr gilt.

Im einfachsten Fall sind die Bedingungen Prae1, Prae2 usw. paarweise disjunkt — dies entspricht dem Umschalten zwischen verschiedenen Betriebszuständen. Es muß jedoch nicht der Fall sein.

Die Funktion des Hubdrehtisches wird mit dem function-Konstrukt wie folgt beschrieben werden:

```
function: [
    part_present ist ja -
        vertical_pos= high ^ rotary_pos= rot 45
    part_present ist nein -
        vertical_pos= low ^ rotary_pos= rot0
]
```

Für jedes Paar Prae — Target wird eine separate Iterationskonstruktion durchgeführt wie unter 4.3. beschrieben. Sie liefert als Ergebnis — wie für die Sicherheit — folgende Informationen

- Anfangszustand okay oder nicht okay, und
- Übergang okay oder nicht okay.



Falls beide Prüfungen "okay" ergeben, erfüllt die resultierende Steuerung per Konstruktion die funktionale Eigenschaft. Andernfalls wird dies nicht garantiert.

#### 4.3 Verfahren zur Funktions-Generierung

In diesem Abschnitt wird der Algorithmus zur Generierung der funktionalen Eigenschaften beschrieben.  
Verwendete Bezeichnungen

— Delta bezeichne die Übergangsrelation der Steuerung. Diese ist zu Anfang vorbelegt mit einer Relation, die vorgegebene Einschränkungen darstellt, wie etwa nach Abschnitt 3 konstruierte Sicherheitseigenschaften.

Im Verlaufe des Verfahrens wird Delta weiter eingeschränkt, und bei erfolgreichem Ende des Verfahrens drückt sie am Schluß gerade die Lösung aus, d. h. die Steuerung, die die verlangte Funktion garantiert.

— Fair<sub>i</sub> bezeichne die Menge, die durch folgende Bedingung gekennzeichnet ist: "der aktuelle Prozeßschritt besteht in einer Aktion der i-ten Prozeßkomponente". Diese Bedingung wird auch als (i-te) Fairness-Bedingung bezeichnet. Hierin spiegelt sich die implizite Voraussetzung, daß jede unabhängige Komponente des Prozesses "immer wieder", d. h. jeweils nach endlicher Zeit aktiv wird. Das Verfahren benutzt diese Annahme in folgender Weise: ein Fortschritt in Richtung auf einen angestrebten Zustand ist auch dadurch erreichbar, daß das Eintreten einer Fairness-Bedingung einen solchen Übergang erzwingt.

— Für jede Zustandsmenge P bezeichne  $acx(P)$  die Menge all der Zustände, für die in einem Schritt ein Übergang in die Menge P erzwingbar ist.

Die größte stabilisierbare Teilmenge von P kann berechnet werden wie unter 3. beschrieben.

Die Bezeichnungen  $win_{n,j}$ ,  $winX_n$ ,  $aux_{n,j}$  stehen für Zustandsmengen, die im Laufe des Verfahrens berechnet werden:

#### Verfahren

Der folgende Algorithmus wird für jedes Paar "Prae — Target" durchgeführt.

- (1) Setze  $win_{n,0}$  := größte stabilisierbare Teilmenge von Target,
- (2) wiederhole die Schritte (3) bis (10) für  $n = 0, 1, 2, \dots$ , solange bis gilt

$win_{n+1,0} = win_{n,0}$

- (3) wiederhole die Schritte (4) und (5) für  $j = 0, 1, 2, \dots$ , solange bis gilt

$win_{n,j+1} = win_{n,j}$

- (4) setze  $win_{n,j+1} := win_{n,j} \cup acx(win_{n,j})$ ; d. h. Die Menge  $win_{n,j+1}$  besteht aus allen Zuständen der Menge  $win_{n,j}$  zuzüglich der Zustände, von denen aus in einem Schritt ein Übergang in die Menge  $win_{n,j}$  erzwingbar ist,

- (5) schränke die Übergangsfunktion Delta in der Weise ein, daß, wenn immer möglich, die Menge  $win_{n,j+1}$  nicht mehr verlassen wird,

- (6) setze  $winX_n := win_{n,j}$

- (7) wiederhole die Schritte (8) und (9) für jede Fairness-Bedingung Fair<sub>i</sub>:

- (8) setze  $aux_{n,j} :=$  größte stabilisierbare Teilmenge von  $(winX_n \cup \text{Menge der Zustände } S, \text{ so daß gilt: falls } S \text{ in Fair}_i \text{ liegt, dann ist jeder Folgezustand in } winX_n)$ ,

- (9) schränke die Übergangsfunktion Delta in der Weise ein, daß, wenn immer möglich, die Menge  $aux_{n,j}$  nicht mehr verlassen wird,

- (10) setze schließlich  $win_{n+1,0} := aux_{n,1} \cup aux_{n,2} \cup \dots \cup aux_{n,k}$ .

#### Erfolgskriterium

Die erzeugte Steuerung ist dann korrekt, wenn am Schluß folgendes gilt:

- (1) Der Anfangszustand des Systems liegt in der Menge  $win_{n,0}$ .
- (2) Für jeden Zustand in  $win_{n,0}$  liegen auch alle Folgezustände in  $win_{n,0}$ .

Im Fig. 6 ist das Verfahren nochmals als Ablaufplan dargestellt.  
Nachfolgend wird ein komplettes Codebeispiel für den Hubdrehstuhl angegeben:

**/\* CSLxt Code für Hubdrehtisch \*/**

**declTyp(rot\_table,**

**sensor\_zustaende: [**

part\_present: [ ja, nein],

vertical\_pos: [ low, between\_low\_high, high ],

rotary\_pos: [ rot0, between\_rot0\_rot45, rot45 ],

**ausgangs\_zustaende: [**

moving\_state: [ stop, up, down ],

rotation\_state: [ rot\_stop, rot\_plus, rot\_minus ],

**process\_description:[**

qual\_deriv :[vertical\_pos, low, between\_low\_high, high,  
moving\_state, stop, up, down],

qual\_deriv :[rotary\_pos, rot0, between\_rot0\_rot45, rot45

rotation\_state, rot\_stop, rot\_plus, rot\_minus, ],

**safety\_requirements:**

**/\*Tisch nicht tiefer als "low" absenken\*/**

nicht (vertical\_pos = low  $\wedge$  moving\_state = down) /\* (etc. fuer die anderen Grenzen) \*/

**/\* Tisch in Stellung "low" nicht rotieren \*/**

$\wedge$  ((vertical\_pos = low)  $\implies$

(rotation\_state = rot\_stop)  $\wedge$  (rotary\_pos = rot0)),

**function:[**

part\_present ist ja -

(vertical\_pos= high)  $\wedge$  (rotary\_pos= rot45),

part\_present ist nein -

(rotary\_pos= rot0)  $\wedge$  (vertical\_pos= low)]

**].**

**/\*----- END TYP rot\_table -----\*/**

Patentansprüche

1. Verfahren zur automatischen Erzeugung einer Steuerung für einen Prozeß,
  - a) bei dem ein nicht deterministischer Automat, der alle physikalisch möglichen Verhaltensweisen der Steuerung beschreibt, festgelegt wird,
  - b) bei dem die erlaubten Zustandsübergänge des von der Steuerung zu beeinflussenden Prozesses beschrieben werden,
  - c) bei dem der Automat so eingestellt wird, daß er vorgegebene Sicherheitsbedingungen erfüllt,
  - d) bei dem der Automat so eingestellt wird, daß er die Funktion des aus Steuerung und Prozeß

bestehenden Systems erfüllt.

2. Verfahren nach Anspruch 1,  
bei dem zur Einstellung der Sicherheitsbedingungen die größte stabilisierbare Teilmenge der als zulässig  
spezifizierten Zustandsmenge iterativ bestimmt wird,

bei dem die Übergangsfunktion des Automaten so eingestellt wird, daß nur jeweils Folgezustände gewählt  
werden, die in dieser größten stabilisierbaren Menge liegen.

3. Verfahren nach Anspruch 2, bei dem zur Einhaltung der Sicherheitsbedingungen für das System aus  
Steuerung und Prozeß folgende Zustände bei der Realisierung ausgeschlossen werden:

jeder Zustand der Steuerung, bei dem alle Folgezustände die Sicherheitsbedingungen nicht erfüllen  
jeder Zustand des Prozesses, bei dem mindestens ein Folgezustand die Sicherheitsbedingungen nicht erfüllt.

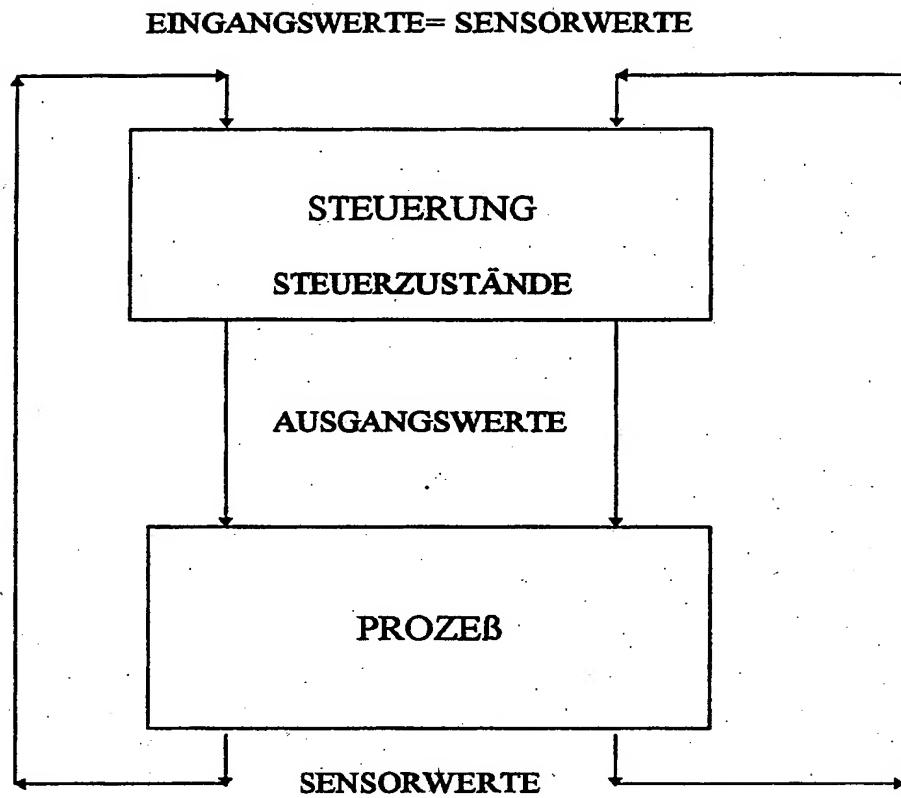
4. Verfahren nach einem der vorhergehenden Ansprüche, bei dem zur Einstellung der Funktion des Systems

— die Funktion durch Angabe bedingter Zielzustände definiert wird,

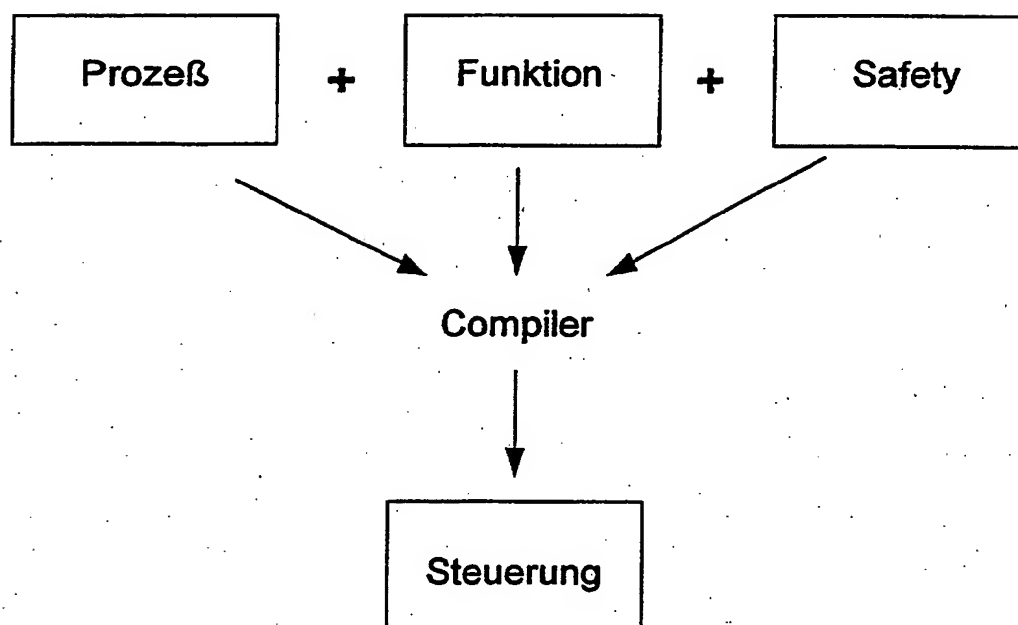
— iterativ die Menge all derjenigen Zustände bestimmt wird, von denen aus der Übergang in einen  
Zielzustand in endlich vielen Schritten erzwungen werden kann,

— die Übergangsfunktion dadurch eingeschränkt wird, daß, wenn möglich, solche Übergänge ausge-  
führt werden, die zum Zielzustand führen.

Hierzu 6 Seite(n) Zeichnungen



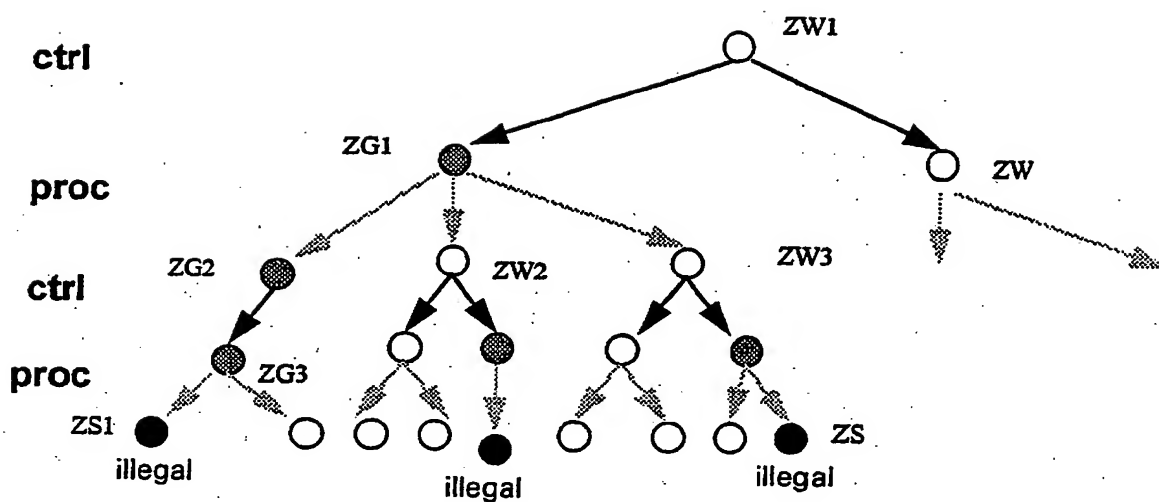
**FIGUR 1**



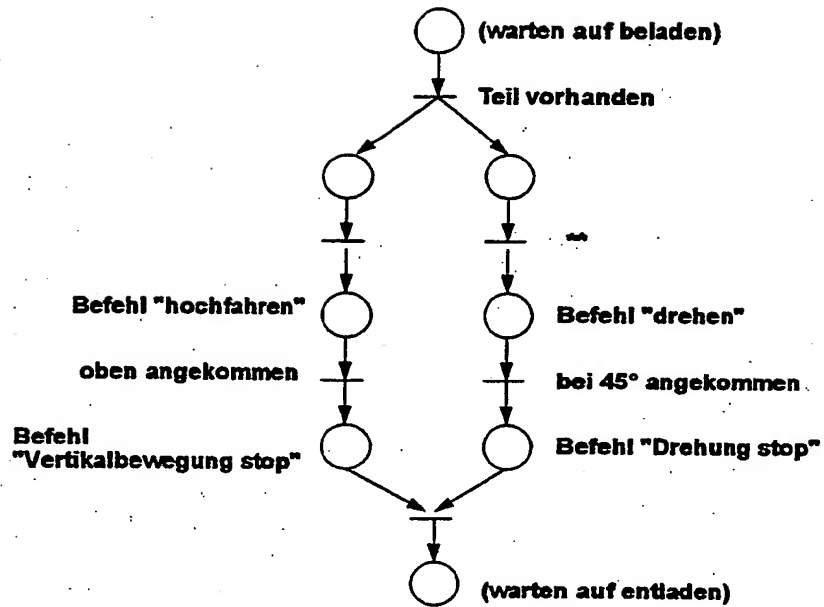
FIGUR 2

high			
between_low_high			
low			
	rot0	between_rot0_rot45	rot45

**FIGUR 3**

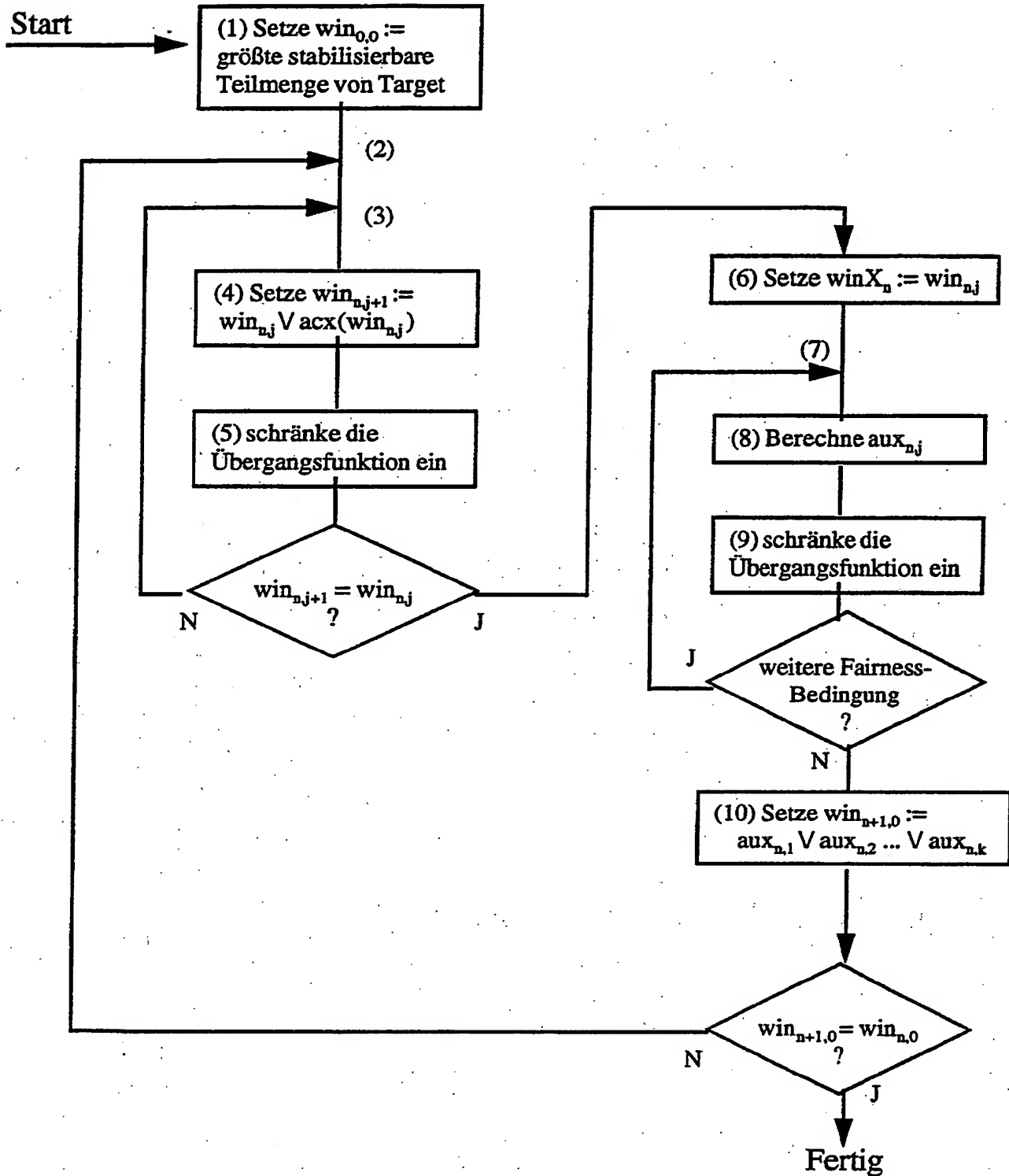


FIGUR 4



FIGUR 5





FIGUR 6

**This Page is Inserted by IFW Indexing and Scanning  
Operations and is not part of the Official Record**

**BEST AVAILABLE IMAGES**

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images include but are not limited to the items checked:

- ☐ **BLACK BORDERS**
- ☐ **IMAGE CUT OFF AT TOP, BOTTOM OR SIDES**
- ☐ **FADED TEXT OR DRAWING**
- ☐ **BLURRED OR ILLEGIBLE TEXT OR DRAWING**
- ☐ **SKEWED/SLANTED IMAGES**
- ☐ **COLOR OR BLACK AND WHITE PHOTOGRAPHS**
- ☐ **GRAY SCALE DOCUMENTS**
- ☐ **LINES OR MARKS ON ORIGINAL DOCUMENT**
- ☐ **REFERENCE(S) OR EXHIBIT(S) SUBMITTED ARE POOR QUALITY**
- ☐ **OTHER:** \_\_\_\_\_

**IMAGES ARE BEST AVAILABLE COPY.**

**As rescanning these documents will not correct the image problems checked, please do not report these problems to the IFW Image Problem Mailbox.**